CENG313 Homework 4 Report

1) Implementation Details

I created **3 mutexes and 2 condition variables** which are respectively queue_mutex, cond_mutex, lock **and** cond_queue, cond_task. Except mutexes and condition variables, I defined **6 external integer variable** which are termination , flag , available_task , synch_btwn_thread_and_task , trapped_thread_count , threads_in_idle.

For purposes and usages of them;

Mutexes:

queue_mutex: It allows to enter threads which exit from wait state to their critical sections after a thread finished its own work.

cond_mutex: In Terminate() function, If there is no task in task queue (available_task) and still the new task will be generated by main thread (flag), our thread must wait for main thread. After waiting main thread, it allows to pull task from task queue one by one.

lock: It is used for ordinary purpose which is that if one thread is in its critical sections, other threads must wait for it thanks to lock.

Contidion Variables:

cond_queue: It waits for signal from main thread until a task is generated. Main thread gives a signal to cond_queue and all threads is woke up one by one.

cond_task: if there is no task in task queue (available_task) and still the new task will be generated by main thread (flag), our thread must wait thanks to cond_task and after main thread generates a new task, it gives a signal to cond_task and thread continues to pull task from task queue.

External Variables:

termination: Main thread controls whether all threads finish its job, after than it prints final list.

flag: If there is still task that will be generated by main thread, flag becomes 1 – otherwise it becomes 0.

available_task: It represents last situation of number of available(generated) tasks.

synch_btwn_thread_and_task: If any thread can not enter its critical section after all tasks are created and done, signal for condition wait is triggered by looking this variable.

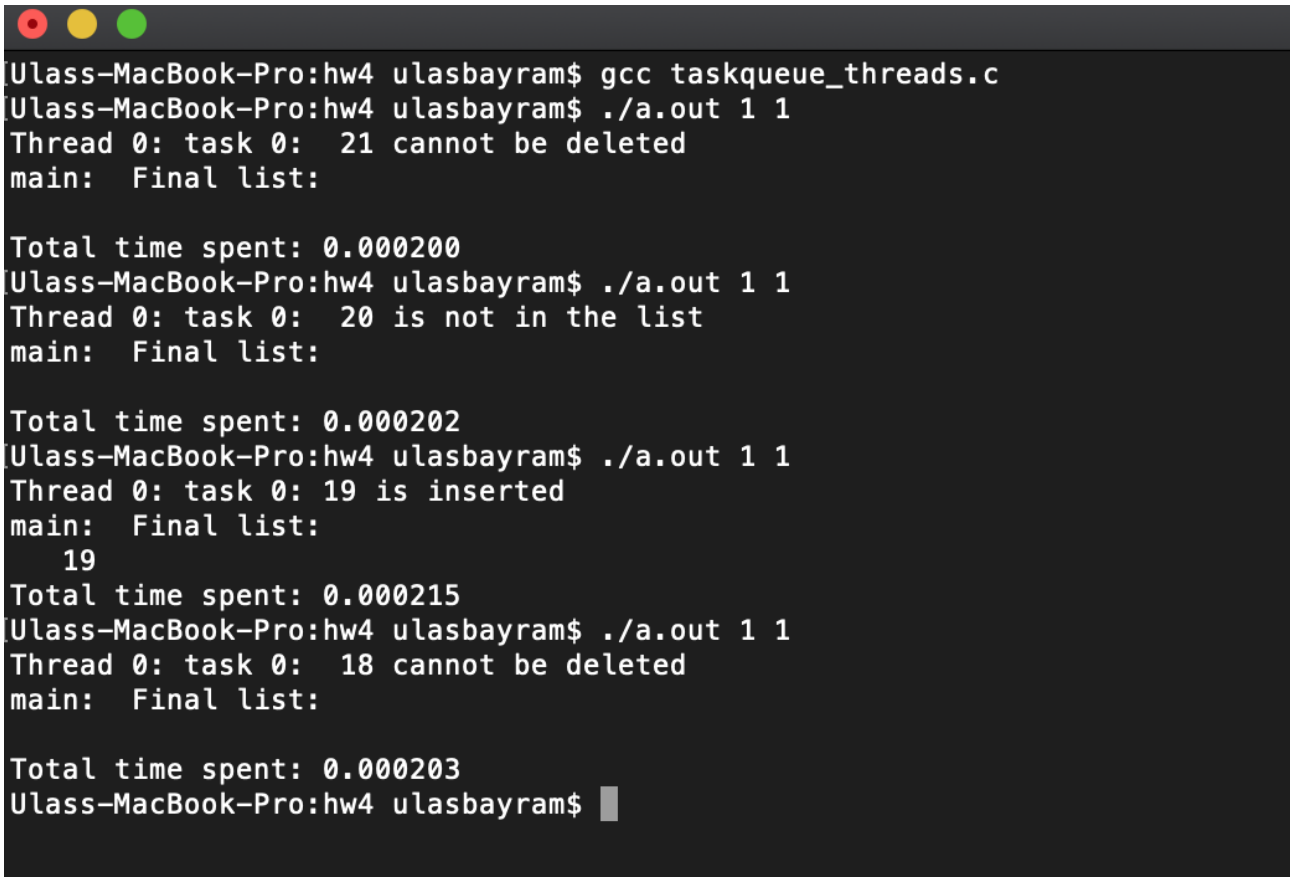
trapped_thread_count: It checks whether number of threads which finish its job is equal to total number of threads or not.

threads_in_idle: After all tasks are generated, if there are any thread that can not enter its critical section, main thread awakens them.

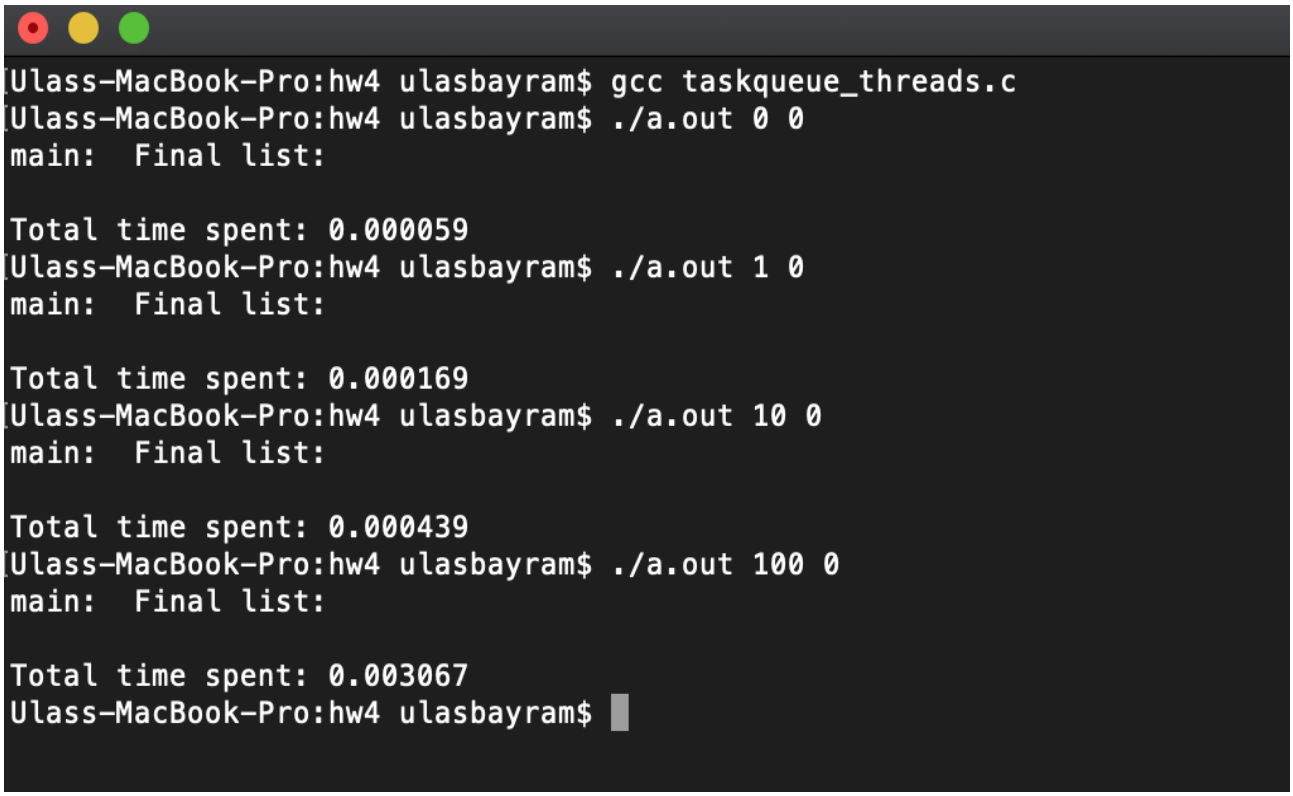
When the main thread generates new task, It allows a thread to pull task from task queue by giving signal to Thread_work() function. The thread pulls a task and finishes its job. When all tasks are generated, all threads are woke up by using broadcast and enter to its critical sections. After all threads finish their executions, main list is printed.

2) Screenshots

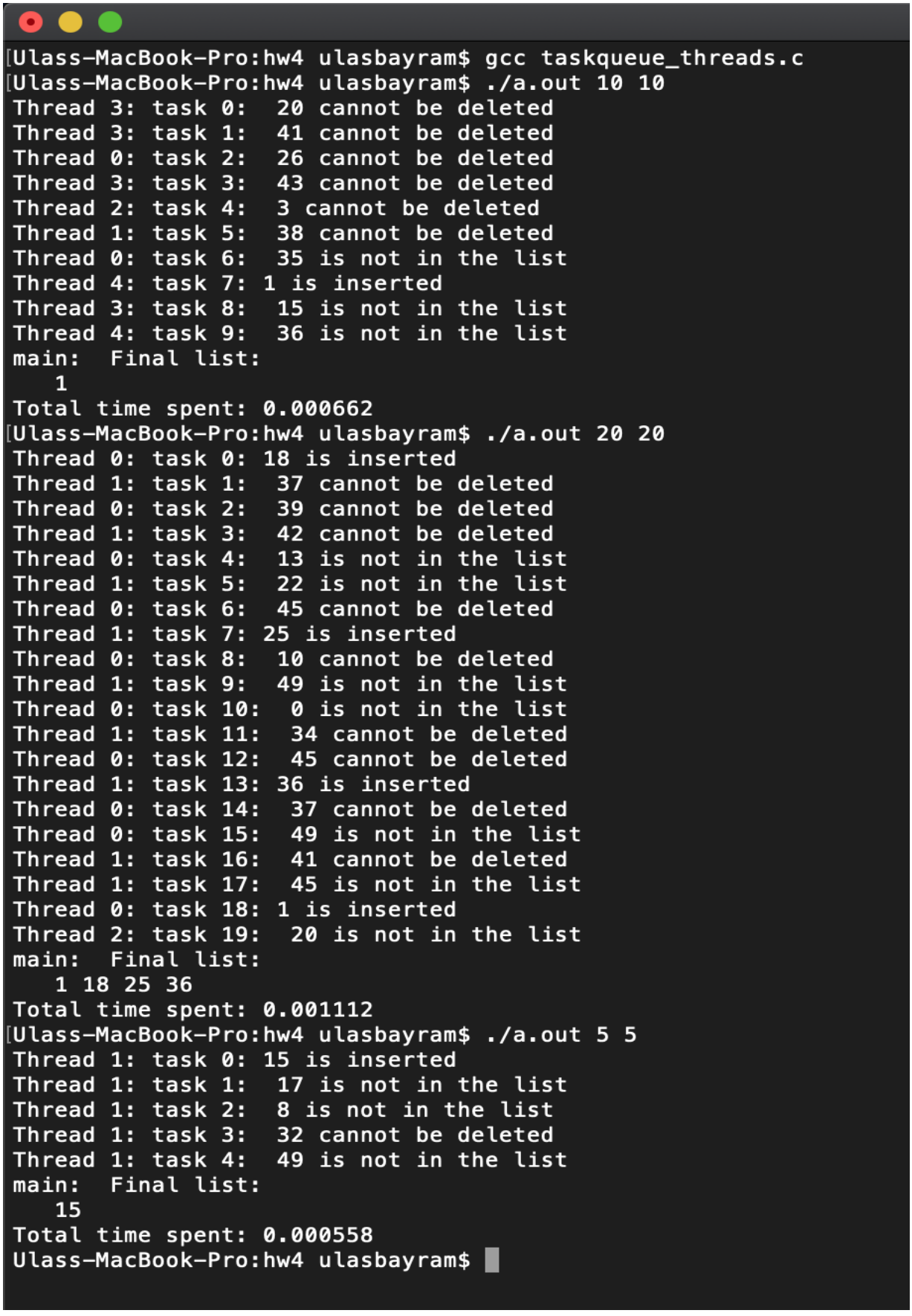
When number of threads and number of tasks are equal to 1;



When number of threads increases and number of tasks is equal to 0;



When number of threads is equal to number of tasks;



When number of tasks increases and number of tasks is equal to static number;

```
Ulass-MacBook-Pro:hw4 ulasbayram$ gcc taskqueue_threads.c
Ulass-MacBook-Pro:hw4 ulasbayram$ ./a.out 2 10
Thread 0: task 0: 26 is inserted
Thread 0: task 1:  19 cannot be deleted
Thread 0: task 2:  45 is not in the list
Thread 0: task 3:  44 cannot be deleted
Thread 0: task 4:  16 cannot be deleted
Thread 0: task 5: 37 is inserted
Thread 0: task 6:  34 cannot be deleted
Thread 0: task 7:  33 cannot be deleted
Thread 0: task 8: 41 is inserted
Thread 0: task 9: 28 is inserted
main:  Final list:
   26 28 37 41
Total time spent: 0.000297
Ulass-MacBook-Pro:hw4 ulasbayram$ ./a.out 5 10
Thread 0: task 0: 23 is inserted
Thread 0: task 1:  49 cannot be deleted
Thread 1: task 2:  14 cannot be deleted
Thread 0: task 3:  34 cannot be deleted
Thread 3: task 4:  2 is not in the list
Thread 0: task 5:  35 is not in the list
Thread 0: task 6:  10 cannot be deleted
Thread 4: task 7:  48 cannot be deleted
Thread 4: task 8:  28 cannot be deleted
Thread 4: task 9:  14 is not in the list
main:  Final list:
   23
Total time spent: 0.000485
Ulass-MacBook-Pro:hw4 ulasbayram$ ./a.out 20 10
Thread 0: task 0: 23 is inserted
Thread 0: task 1:  26 is not in the list
Thread 1: task 2:  30 is not in the list
Thread 0: task 3:  24 cannot be deleted
Thread 3: task 4: 38 is inserted
Thread 2: task 5:  36 cannot be deleted
Thread 1: task 6:  36 is not in the list
Thread 5: task 7:  10 cannot be deleted
Thread 8: task 8:  15 is not in the list
Thread 5: task 9: 0 is inserted
main:  Final list:
   0 23 38
Total time spent: 0.001067
Ulass-MacBook-Pro:hw4 ulasbayram$
```

When number of tasks increases and number of tasks is equal to static number – 2;

```
Ulass-MacBook-Pro:hw4 ulasbayram$ gcc taskqueue_threads.c
Ulass-MacBook-Pro:hw4 ulasbayram$ ./a.out 50 10
Thread 3: task 0:  10 cannot be deleted
Thread 3: task 1:  5 is not in the list
Thread 0: task 2:  10 is not in the list
Thread 3: task 3: 45 is inserted
Thread 1: task 4:  13 is not in the list
Thread 2: task 5: 1 is inserted
Thread 0: task 6:  37 cannot be deleted
Thread 2: task 7: 47 is inserted
Thread 3: task 8: 16 is inserted
Thread 3: task 9: 34 is inserted
main:  Final list:
   1 16 34 45 47
Total time spent: 0.002032
Ulass-MacBook-Pro:hw4 ulasbayram$ ./a.out 100 10
Thread 2: task 0:  7 cannot be deleted
Thread 2: task 1: 9 is inserted
Thread 1: task 2:  48 is not in the list
Thread 2: task 3: 25 is inserted
Thread 4: task 4: 32 is inserted
Thread 4: task 5: 0 is inserted
Thread 7: task 6:  39 is not in the list
Thread 6: task 7:  24 is not in the list
Thread 7: task 8:  40 is not in the list
Thread 6: task 9:  6 cannot be deleted
main:  Final list:
   0 9 25 32
Total time spent: 0.003608
Ulass-MacBook-Pro:hw4 ulasbayram$ ./a.out 500 10
Thread 2: task 0: 2 is inserted
Thread 2: task 1:  8 cannot be deleted
Thread 2: task 2:  11 cannot be deleted
Thread 4: task 3: 24 is inserted
Thread 2: task 4:  42 cannot be deleted
Thread 0: task 5:  34 is not in the list
Thread 0: task 6:  49 cannot be deleted
Thread 3: task 7:  48 is not in the list
Thread 6: task 8: 35 is inserted
Thread 6: task 9:  16 cannot be deleted
main:  Final list:
   2 24 35
Total time spent: 0.016058
Ulass-MacBook-Pro:hw4 ulasbayram$
```

3) Graphs

For 1000 Tasks;

Sequential Version -> 0.003938 second

Multithreading Version: 1 -> 0.005011 second || Speedup: 0.785871

Multithreading Version: 2 -> 0.004740 second || Speedup: 0.830801

Multithreading Version: 4 -> 0.004313 second || Speedup: 0.913053

Multithreading Version: 8 -> 0.004231 second || Speedup: 0.930749

Multithreading Version: 16 -> 0.004915 second || Speedup: 0.801220

For 10000 Tasks;

Sequential Version -> 0.042035 second

Multithreading Version: 1 -> 0.047526 second || Speedup: 0.884463

Multithreading Version: 2 -> 0.062118 second || Speedup: 0.676695

Multithreading Version: 4 -> 0.074692 second || Speedup: 0.562777

Multithreading Version: 8 -> 0.077521 second || Speedup: 0.542240

Multithreading Version: 16 -> 0.078058 second || Speedup: 0.538509

For 100000 Tasks;

Sequential Version -> 0.402423 second

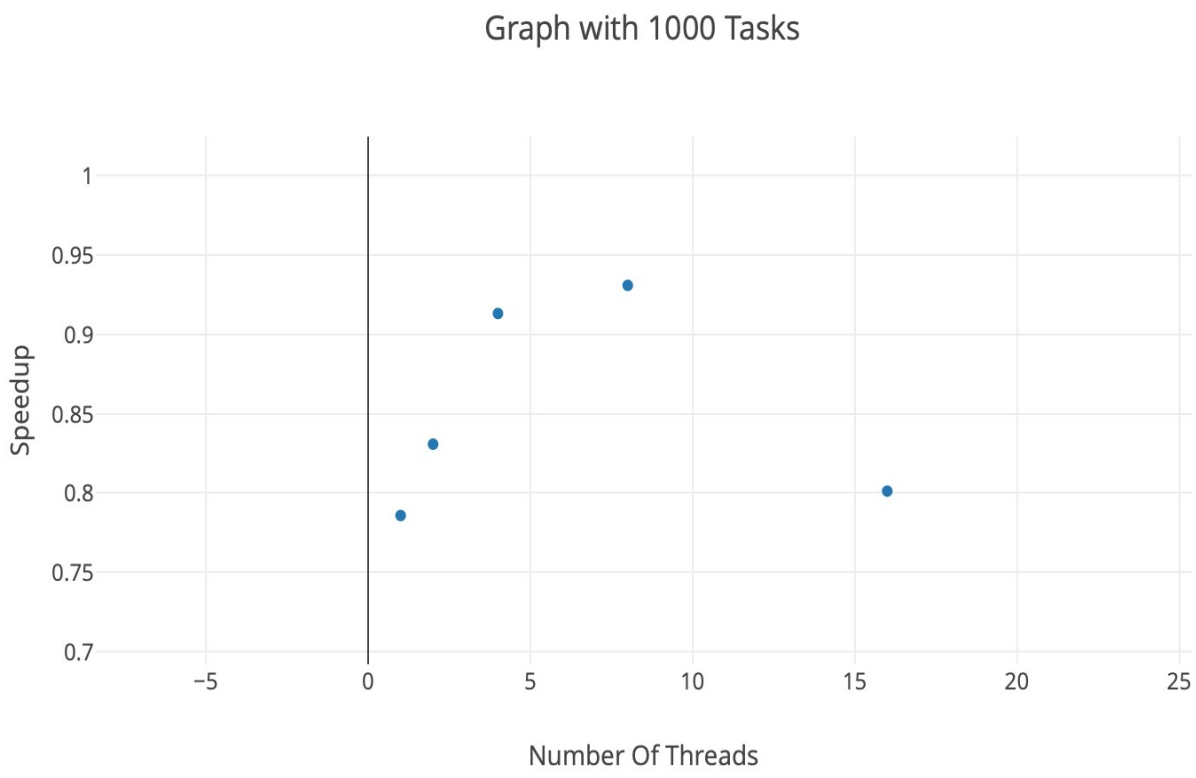
Multithreading Version: 1 -> 0.422686 second || Speedup: 0.952061

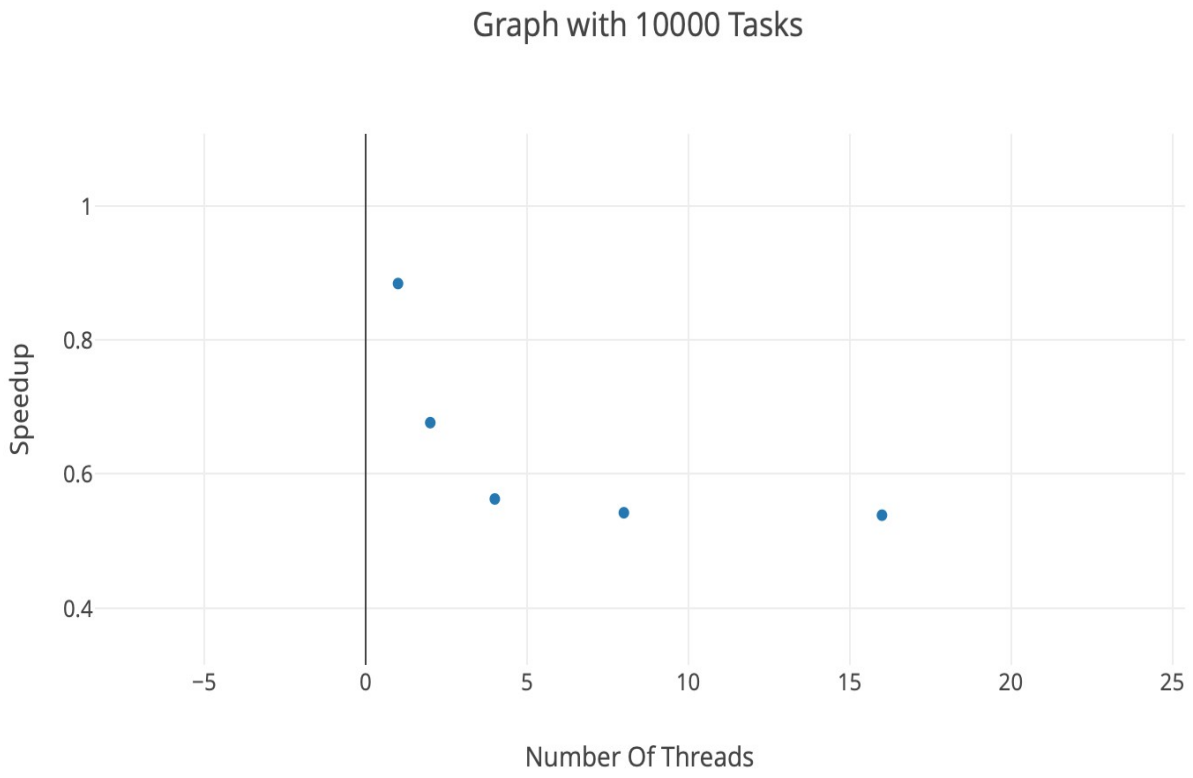
Multithreading Version: 2 -> 0.568046 second || Speedup: 0.708433

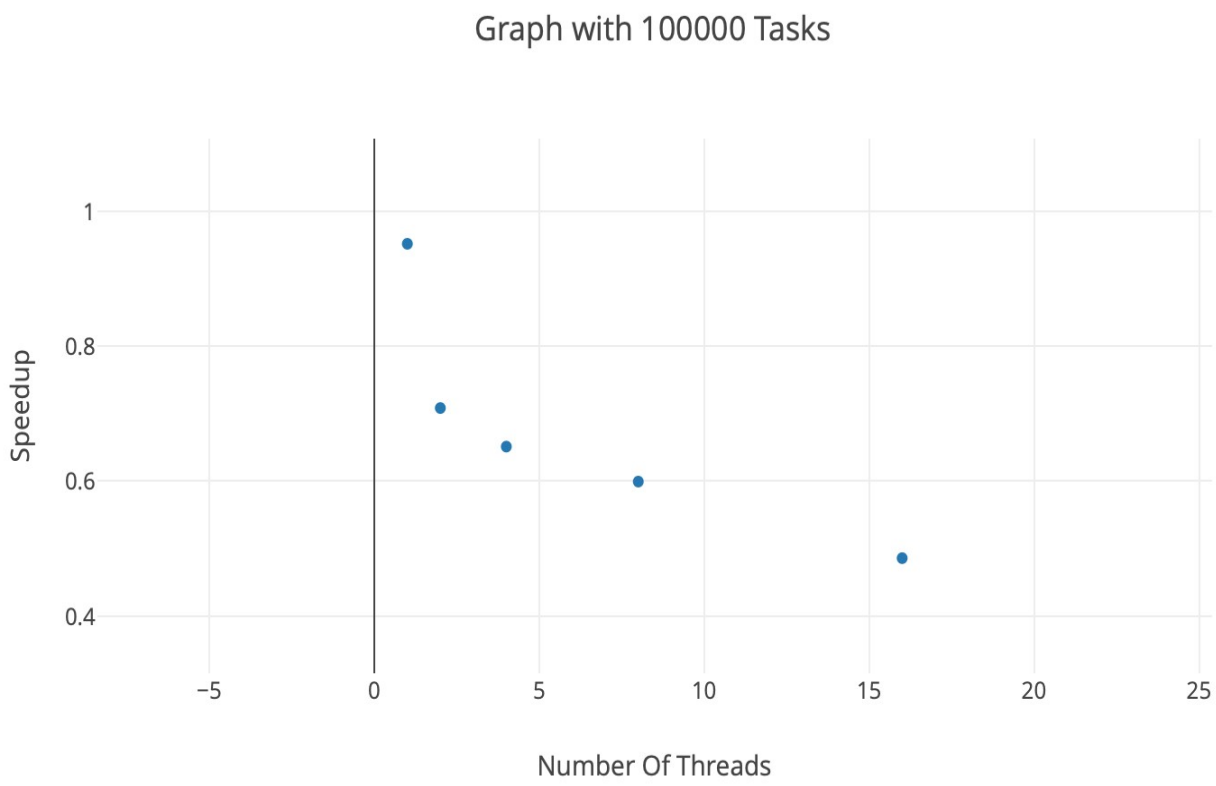
Multithreading Version: 4 -> 0.617959 second || Speedup: 0.651213

Multithreading Version: 8 -> 0.671501 second || Speedup: 0.599288

Multithreading Version: 16 -> 0.828570 second || Speedup: 0.485683




Graph with 10000 Tasks
Speedup
1
0.8
0.6
0.4
−5
0
5
10
15
20
25
Number Of Threads


Graph with 100000 Tasks
Speedup
1
0.8
0.6
0.4
−5
0
5
10
15
20
25
Number Of Threads

4) Observations About The Performance Of Your Implementation

By looking these graph, all conditions are different from each others. These conditions are context switch, number of tasks, I/O operations or whether CPU is busy or not. We can not make any interpretation about CPU busy time because it depends on how busy the processor is at that moment. When examining first graph, threads work concurrently and have small number of context switch that affects time cost directly. Since number of tasks is smaller than others for first graph, it has less context switch time and I/O operation(Interrupt) time. For second and third graphs, since number of tasks are greater than first one, context switch and I/O operation time are grater than it. We can make a deduction like this; when looking context switch and I/O operation's costs, increasing number of threads does not give best result every time.

Ismail Ulaş Bayram
220201040